Background Information

The Calculators

The TI-84 calculator is a graphing calculator made by Texas Instruments for high school and college students. First introduced in 1999, even then the calculator was outdated technology. Based around the 1970s 8-bit microprocessor, the Zilog Z80, the TI-84 has a 96x128 monochrome display, 24 kB of user accessible RAM, and an 8 MHz processor. The operating systems most unique feature is the built in programming language, editor, and interpreter known as TI-BASIC (although it has no official name). This language, although not a true BASIC (Beginner's All Purpose Instruction Code) shares many similarities, but is missing a lot of important functionality that even the most basic structured programming languages contain such as user defined functions, multiple numeric types, and variable scoping to name a few. TI-BASIC is also deficient in its access to the calculator hardware, bitwise operations, and finite set of inadequate identifiers. Even with its defects, many wonderful programs from symbolic integrators to chess games are written in TI-BASIC. It is important to note that all comments on the TI-84 apply equally to the TI-83 and all variations in the TI-83 Family.

Compilers vs Interpreters

Classical Computers operate by reading and executing very simple instructions called machine code. However, the binary 1s and 0s of machine code are nearly impossible to read and write directly, so higher level languages were developed to abstract over machine code. Two strategies for executing high level languages are to compile then execute them, or to interpret them. Compilers take high level language source code such as C or Java code and compiles it down into machine code that accomplishes the same task as the high level code. Then the compiled code can be executed by the computer. An interpreter, on the other hand, reads the high level source code such as Javascript or TI-BASIC line by line and executes each high level command one by one. Interpreted language implementations are usually much slower than compiled languages because machine code operates very quickly, while the interpreter takes longer to read, interpret, and execute instructions. This is one of the largest drawbacks of TI-BASIC, but it could easily be addressed by writing a compiler for it.

Engineering Goal

The goal for this engineering project is to create a compiler to translate a new programming language based on TI-BASIC into machine to be run on a TI-84 calculator. This compiler would first be implemented in Java, but eventually run on the calculator as a flash application and take as input the name of a BASIC program and create an equivalent assembly program on the calculator. The goal of the project is that the generated machine code when run will have the same result as the interpreted code, but run in much less time. The goal is to support every BASIC instruction and mathematical operation, but the highest priority functionality is that which deals with arithmetic, control flow, and I/O with the keypad and home screen. Another goal of the project is to improve BASIC by adding features such as functions, variable scope, improved I/O capabilities, and better numeric types.

Procedure

1. Determine what features of TI-BASIC to keep and what features to add
2. Write test programs that will exercise selected functionality
3. Write basic lexer, parser, z80 assembly library, and code generator
4. Hook up assembly code output from code generator to ORG assembler
5. Implement language features incrementally in the lexer
6. Implement language features incrementally in the parser
7. Implement language features incrementally in the z80 assembly library
8. Implement language features incrementally in the code generator
9. Test newly added language features using test code created in step 2
10. Evaluate progress and adjust project parameters. Iterate back to step 5

Implemented Language Features

Types:
- 32 Bit Integers
- Lists
- Strings

Input / Output:
- Disp
- Output
- Pause
- getKey
- ClrHome

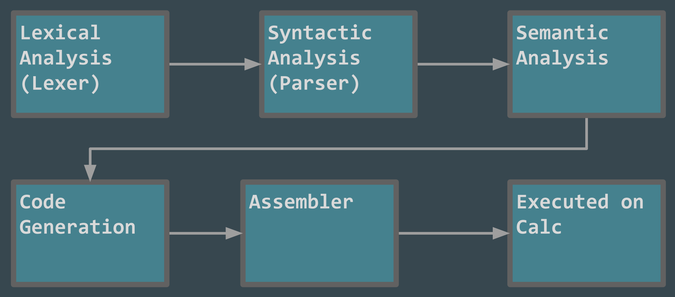
Operations:
- Numeric Operations
  - Arithmetic (+,-,*,/)
  - Modulus
  - Absolute value
  - randInt
- List Operations:
  - Augment
  - Dim
  - Pop
- String Operations
  - Concatenation
  - Num2Str
- Store (assignment)
- Logical Operators
  - Not
  - And
  - Or
- Bitwise Operators
  - Bitwise logic (&,|,^)
  - Left Logical Shift <<
  - Right Logical Shift >>
- Comparison Operators
  - String =
  - Integer =
  - <, <=
  - >, >=

Control Structures:
- Functions
- Stop
- While
- Repeat
- For
- Return
- If
- Else
- ElseIf

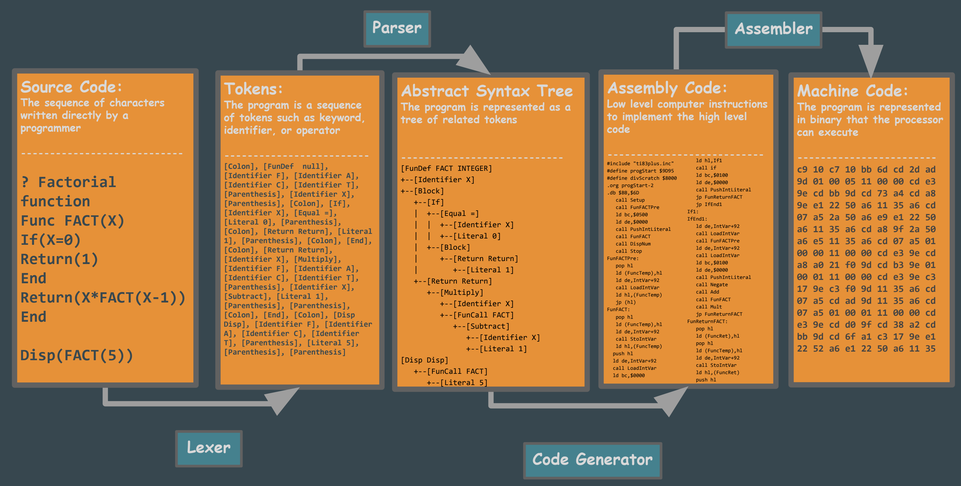
Miscellaneous:
- Hexadecimal Literals
- Single Line Comments
- On Break
- Integer to String conversion
- getTime
- Variable Scope

New features are underlined in orangee.

Compiler Diagram



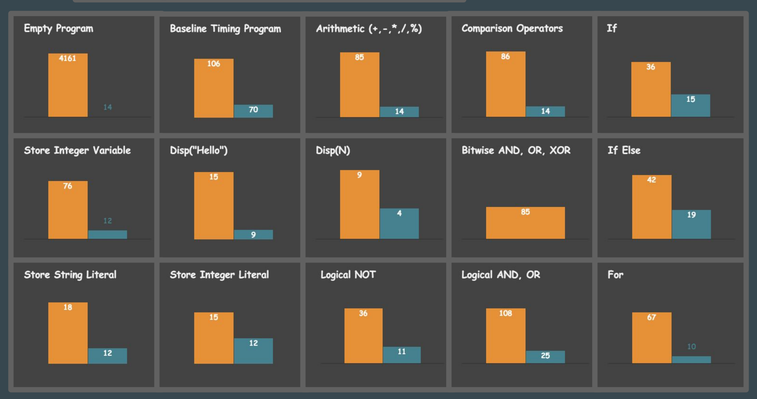
Intermediate program representations



Development Tools & Workflow

- Apache Ant
  - Ant is a Java build tool that automates compilation, creation of an executable, and other software development tasks
- Java Runtime Environment and Java Development Kit
  - Java tools for compiling and running java applications
- jsTIfied
  - Online TI-84 emulator used for testing compiled <NAME> code
- ClrHome ORG Assembler IDE
  - Z80 Assembly IDE accessed through its web API to create machine code from assembly
- Vim
  - An efficient text editor for programmers
- Git
  - A version control tool accessed through the command line interface for managing changes to a project repository
- Github
  - An online tool for hosting git repositories allowing remote access from different developers and machines
- Trello
  - A virtual Kanban board used to organize task logs and track project progress

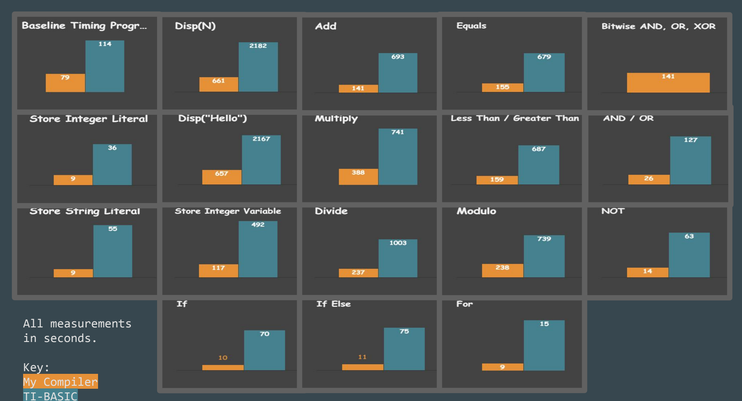
Compiled Memory Data



All measurements in bytes. Key: My Compiler TI-BASIC

All measurements are deltas from baseline timing program (Baseline Timing program is a delta from empty program)

Runtime Speed Data



All measurements in seconds.

Key:
My Compiler
TI-BASIC

Results

While the compiler still has room for improvement in both size and speed of compiled code, it certainly proved the viability of compiling TI-BASIC and made major improvements on the language. While the compiler is a step towards a faster programming language for TI-84 calculators, it was not as fast as initially anticipated. Even so, it is significantly faster than TI-BASIC . With optimizations, I am optimistic that further improvements to speed can be achieved to be orders of magnitude faster than TI-BASIC. Size however, is more difficult to optimize for and TI-BASIC's use of tokens and an interpreter makes it difficult to beat in terms of size. With optimizations to reduce size and introduction of higher level language features, I believe that program size can be reduced to levels similar to TI-BASIC or even smaller. You'll notice that in the data there are several measurements where TI-BASIC has no results. This is because it does not support that feature, showing the compiler has been successful at making important improvements to TI-BASIC.

Conclusions

I think that this project can be useful for existing TI-BASIC programmers, z80 assembly programmers, and even the average calculator user. TI-BASIC programmers will benefit from the higher level language features, while z80 assembly programmers benefit from the possibility of increased speed and control over the hardware.

- Benefits the community of TI-84 programmers and average calculator users.
- Allows new interesting and useful applications to be built, such as an enjoyable chess program or a utilitarian symbolic integrator.
- Provides a good gateway to introduce new programmers to calculator programming
- Aligns more closely with existing programming languages like C and Java.
- Provided as open-source project on Github, allowing features to be added to the language by user community.

Future Work

This project, and engineering projects in general, is never finished and there is still a lot of work to be done on it:

- Optimize the compiler for both speed and size, possibly
- Implement additional features of BASIC (floating point numbers, advanced mathematical operations, user input function, etc.)
- Continue developing new features (Error handling, a more advanced type system, interop with BASIC and z80 assembly, etc.)
- Implement the compiler in z80 assembly on the calculator for mobile programming.
- Integrate the language with the operating system to fully replace TI-BASIC